FIUBA - 75.07

Algoritmos y programación III

Trabajo práctico 2: Al-Go-Oh!

1er cuatrimestre, 2018

(trabajo grupal de 4 integrantes)

Alumnos:

Nombre	Padrón	Mail
Andrés Manuel Blanco	99246	andublanco@yahoo.com.ar
Fernando Gabriel Alvarez	99373	feeerr_95@hotmail.com
Florencia Villar	99353	florvillar@outlook.com
Juan Alvarez Windey	95242	juan.alvarezwindey@gmail.com

Informe

Introducción

Desarrollamos nuestro trabajo práctico utilizando la metodología TDD.
Examinamos los puntos a desarrollar y realizamos pruebas de integración concordando con lo requerido. Una vez escrito el código debimos actualizar nuestros diagramas y realizar más pruebas unitarias y de integración para ver que todo funcione tal lo esperado. Una vez que completamos lo que nos pidieron con respecto a la funcionalidad del juego, realizamos la interfaz gráfica del trabajo.
La idea de este trabajo es realizar la aplicación del juego Yu-Gi-Oh adaptado como Al-Go-Oh.

Supuestos

- En el juego real, el jugador tiene la posibilidad de decidir cuando quiere activar su carta trampa. En nuestra versión, la carta trampa se activa automaticamente cuando el jugador es atacado por su oponente.
- Cuando se va a colocar una carta que requiere sacrificar una o más cartas, las cartas que van a ser sacrificadas (y por ende, destruidas), son elegidas al azar.
- Si se activa una carta trampa cuyo efecto es destruir a un monstruo enemigo, ese monstruo enemigo es elegido al azar.
- En el primer turno se pueden colocar más de una carta.

Modelo de dominio

Decidimos hacer dos bases de datos: una de las cartas, que contiene todas aquellas que pueden ser utilizadas en una partida, y una de efectos, que contiene una clase por cada efecto de cada carta mágica, trampa, de campo y monstruo.
Implementamos dos interfaces: Afectable y ZonaDeJuego. La primera consiste en un método que se llama recibirEfecto, que esta implementado en las clases Carta, Jugador y Tablero, ya que son los tres elementos del juego a los que se les puede aplicar un efecto. Cada una de estas clases implementa la interfaz Afectable y tiene el método recibirEfecto implementado según corresponda.
Además implementamos una clase llamada Tablero que tiene dos atributos Campo, cada unos correspondiente a un jugador. Esta clase Campo es la que contiene a todos los elementos del juego que fueron creados usando clases: las dos zonas, ZonaAtaque y ZonaDefensa, el cementerio y la mano que los implementamos como ArrayLists, el mazo, para el cual creamos una clase Mazo y además tiene como atributo al Tablero.
Un jugador iniciará el juego teniendo su mano, que consta de cinco cartas, y agarrando una carta del mazo, colocará las cartas como crea correspondiente y pasará su turno. El oponente hará lo mismo, hasta que alguno de los dos logre quitarle todos los puntos de vida al otro, momento en el que el juego termina y el que no tiene más puntos de vida pierde.
Tenemos una clase abstracta Carta que tiene los siguientes atributos: nombre, posicion, dueño, estado, cara y efecto. Tiene además un método abstracto que se llama agregarseAlCampo y esta definido en clases que heredan de Carta: CartaEfecto, CartaMonstruo y CartaTerreno, ya que cada una de estas cartas va colocada en una zona especial del Campo. CartaMonstruo tiene todos los atributos de un monstruo: puntos de ataque, puntos de defenza, nivel y el nombre. En esta clase esta implementado un método muy importante de la aplicación: atacarOtraCarta. Este método contiene toda la lógica para realizar un ataque entre dos cartas, que es el modo principal de sacarle puntos de vida al oponente y, consecuentemente, ganar el juego.
Luego, de CartaEfecto heredan las clases CartaMagica y CartaTrampa, y de ellas heredan todas las cartas mágicas y de trampa implementadas. Cada carta magica y de trampa tiene a su efecto, que es una clase decladara en el paquete BaseDatosEfectos.

Detalles de implementación

Uno de los puntos conflictivos que tuvimos fueron los efectos. Terminamos implementando una interfaz Afectable con un método que se llama recibirEfecto, que esta implementado en las clases Carta, Jugador y Tablero, ya que son los tres elementos del juego a los que se les puede aplicar un efecto. Cada una de estas clases implementa la interfaz Afectable y tiene el método recibirEfecto implementado según corresponda. Además, otro problema que tuvimos fueron las cartas trampa, ya que inicialmente ibamos a hacer que el jugador pudiera elegir cuándo activarlas, pero terminamos haciendo que se activaran ante el primer ataque del oponente. Frente a esto tuvimos que modificar el metodo de la clase CartaMonstruo atacarAOtraCarta para que preguntara si el jugador atacado tiene cartas trampa en su campo y en el caso de que las tenga, que las active, llamando al efecto correspondiente.
Inicialmente habíamos pensado implementar a las distintas zonas del campo como ArrayLists de tamaño fijo, ya que en el juego no se pueden tener más de cinco monstruos ni más de cinco cartas mágicas/trampa en el campo al mismo tiempo. Al final lo terminamos

implementando como clases separadas que heredan de una interfaz llamada ZonaDeJuego. Cada una de las zonas tiene un ArrayList donde están las cartas y además un diccionario llamado cartasEnJuego que tiene como clave el nombre de la carta y como valor la lista de cartas en juego. Esto está hecho así para que cuando se necesita buscar una carta en el campo, se compare el nombre de la carta con cada una de las cinco o menos cartas en juego y sse obtenga el objeto Carta.
Otro punto conflictivo del trabajo fueron las posiciones en las que podia estar la carta cuando fuera jugada. Inicialmente le pasabamos una cadena que era la abreviación de la posicion al campo, y al final terminamos haciendo que cuando la carta se coloca en el campo, si es un monstruo, se puede elegir en que posicion va a estar, y al método agregarCarta de la clase Campo hay que pasarle por parámetro un objeto PosicionVertical o PosiciónHorizontal y un objeto BocaArriba o BocaAbajo. Estos objetos que refieren a la posición de la carta se crean cuando se llama al método agregarCarta.

Excepciones

Implementamos las siguientes excepciones:

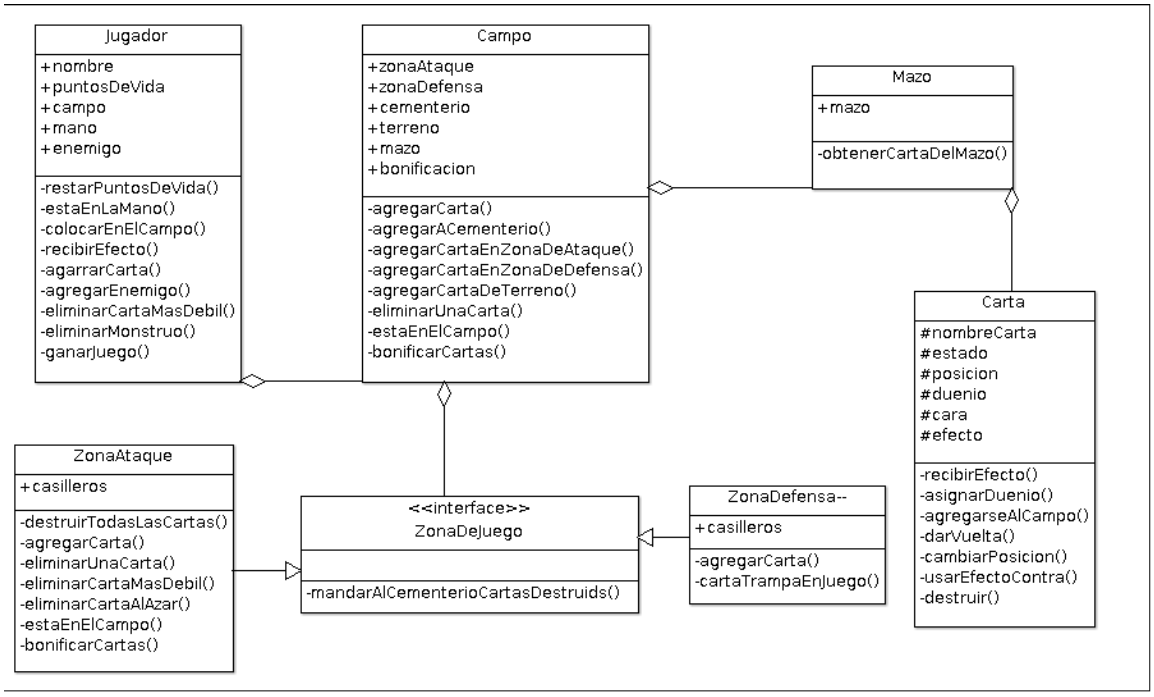
NoSeEncuentraLaCarta:
Cada vez que buscamos una carta no existente, ya sea por confundir el nombre o pasar otra cosa por parámetro, lanzamos esta excepción.

NoHayMonstruosEnCampo:
Implementamos esta excepción para ser lanzada cuando en el campo del jugador correspondiente no haya cartas monstruo y esto sea lo requerido por el atacante.

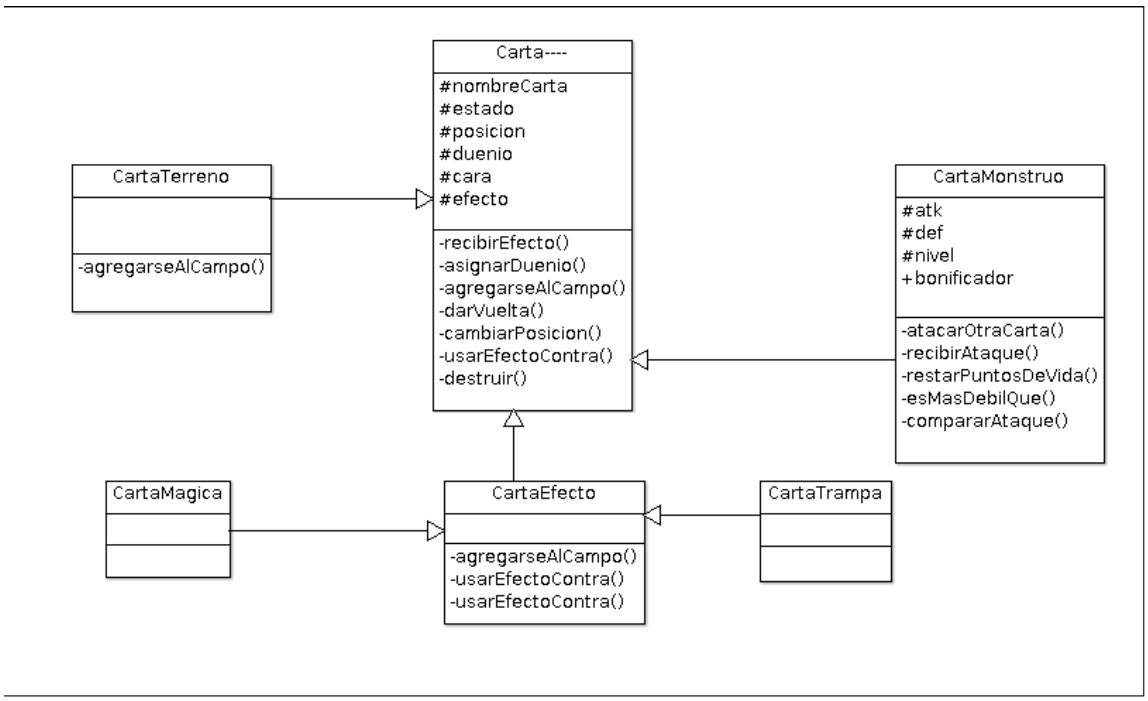
InsuficienteEspacioEnCampo:
Cuando el campo de un jugador está lleno y quiere agregar otra carta se lanzará esta excepción.

Todas ellas heredan de RuntimeException.

Diagramas de clase



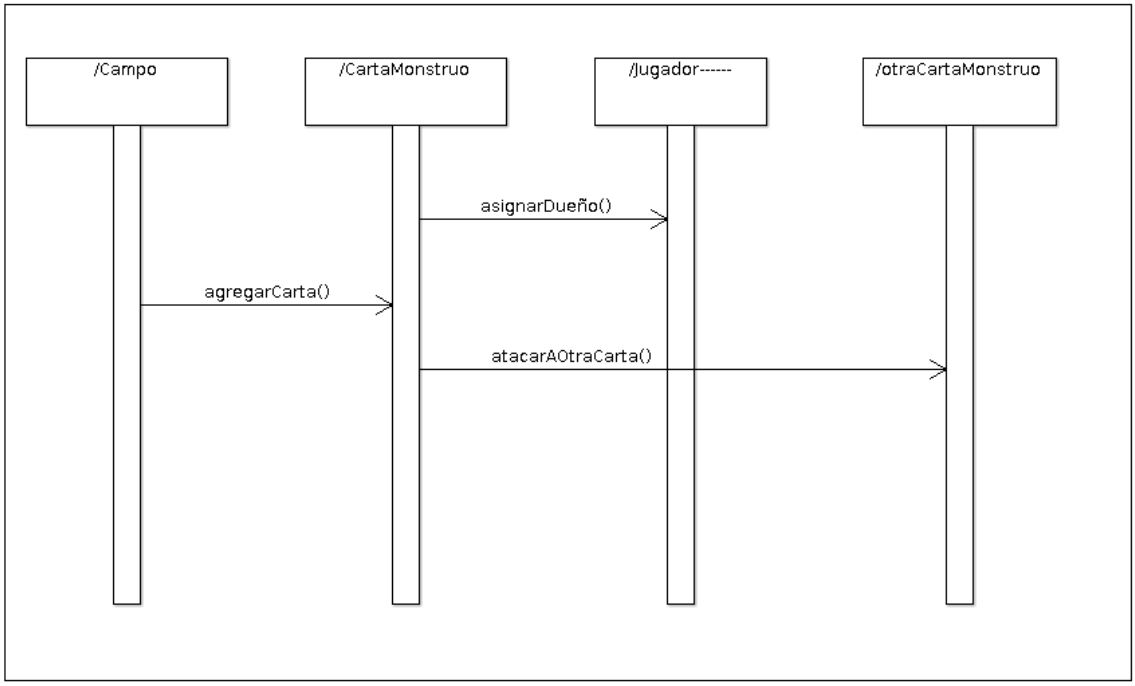
En este diagrama de clases se pueden ver como interactuan las distintas clases principales de nuestro programa.



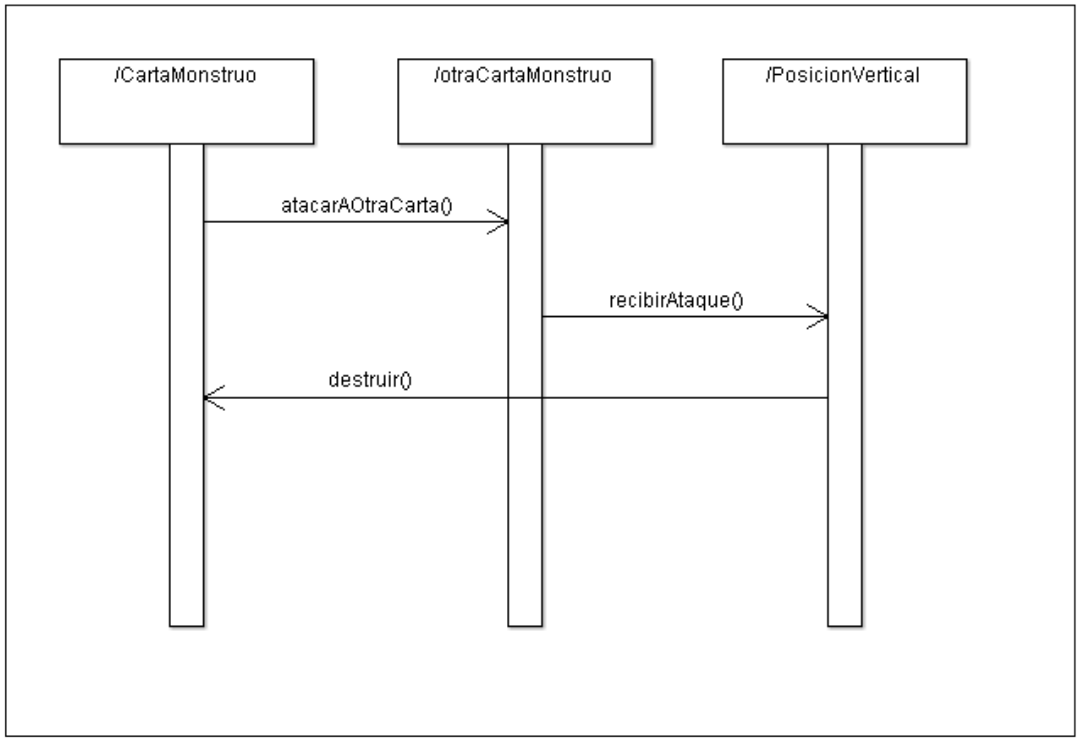
En este diagrama de clases se pueden ver las interacciones entre la clase Carta y sus hijas.

Diagramas de secuencia

Estos diagramas de secuencia representan el ataque de una carta a otra.



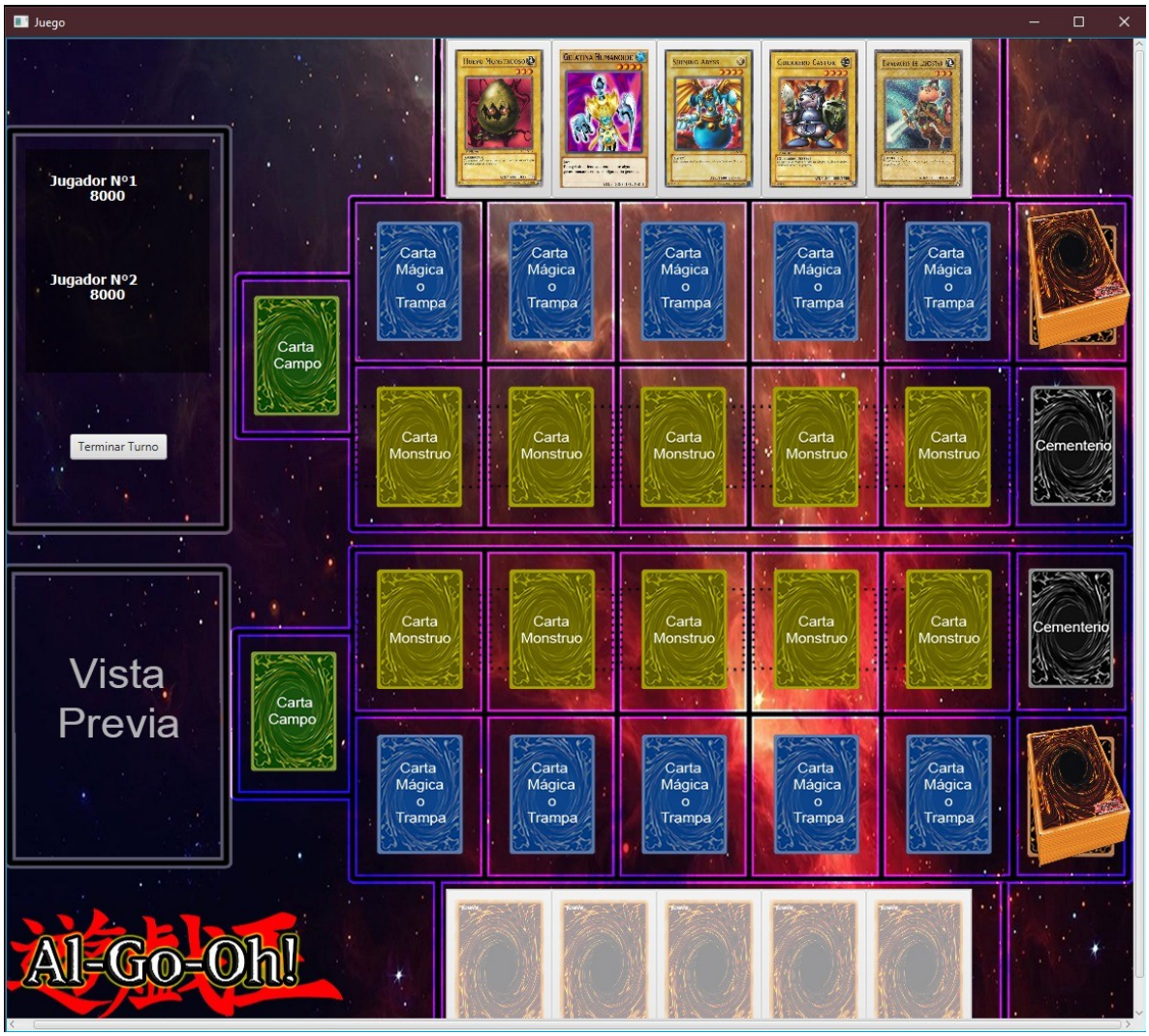
En este diagrama se puede ver como a una carta se le asigna un dueño, es decir, un jugador, y luego esa carta de agrega al campo de ese jugador. Luego, la CartaMonstruo ataca a otraCartaMonstruo. Este diagrama de clases fue hecho a partir de una prueba integradora, donde primero se crean los 4 objetos, luego se le asigna la carta al jugador, luego se agrega la carta al campo, y finalmente se llama al metodo para que la carta ataque.



En este segundo diagrama de secuencia se puede ver como esta implementado el método de CartaMonstruo, atacarAOtraCarta(). La CartaMonstruo le manda un mensaje a otraCartaMonstruo, que luego recibe el ataque y se lo pasa a PosicionVertical, porque la carta atacada (otraCartaMonstruo) estaba en posicion vertical. Luego, la clase PosicionVertical se encarga de comparar los ataques de las dos cartas, y en este caso, determina que los puntos de ataque de la CartaMonstruo son inferiors a los de la otraCartaMonstruo, y que por eso debe ser destruida.

Interfaz grafica

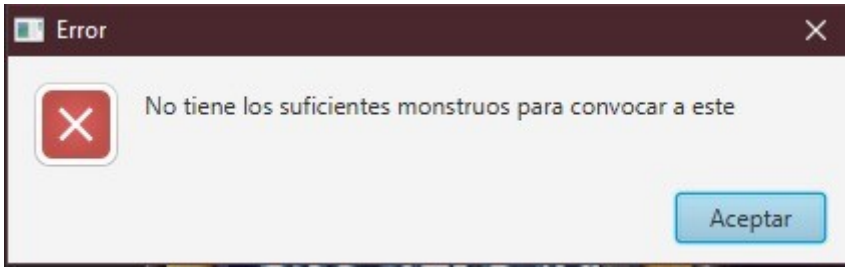
Para realizar la interfaz grafica se utilizó el patrón de arquitectura MVC, que consiste en un controlador, un modelo y una vista. En nuestra aplicación, el controlador manipula al modelo, el modelo actualiza la vista, y la vista se muestra al usuario, que usa el controlador. En la vista estan los eventos de los botones, que se comunican con el controlador, el controlador con esa informacion maneja al modelo, y el modelo actualiza la vista con la informacion que el controlador le paso.



En esta imagen se puede ver el juego funcionando. La partida acaba de arrancar, ninguno de los jugadores perdió puntos de vida, y el el turno del jugador cuya mano se puede ver.

En esta imagen se puede ver que el jugador de la parte superior invocó a un monstruo y paso su turno, y el jugador de la parte inferior invocó a cuatro monstruos, tres en posicion de ataque pero uno solo boca arriba y uno en posicion de defensa boca abajo. El monstruo que está boca arriba es Jinzo 7, cuyo efecto especial le permite atacar al otro jugador directamente a sus puntos de vida.

En esta imagen se puede ver a que cartas puede atacar el jugador superior. En este caso, puede atacar a cualquiera de ellas, pero desconoce que monstruos tiene en su campo el jugador inferior.



Finalmente, este es el mensaje de error que apareció cuando el jugador superior intentó invocar a un monstruo que requería dos sacrificios, teniendo un solo monstruo en el campo.